

WRDC-TR-90-8007
Volume V
Part 8



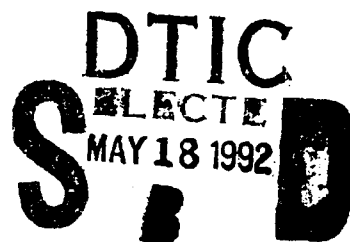
AD-A250 450



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 8 - Neutral Data Manipulation Language (NDML) Reference
Manual

J. Althoff, M. Apicella

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

92-12867



MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

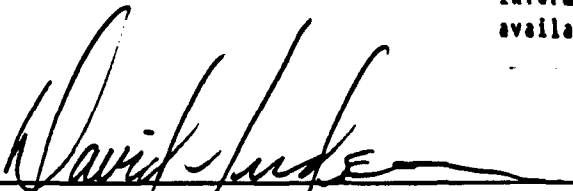
92 5 1

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

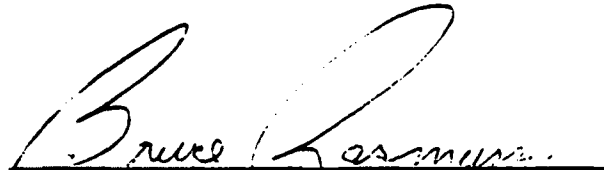
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations


DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:


BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) PRM620341200		5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. V, Part 8	
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services	6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI	
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209		7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF	8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464	
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See block 19 Reference Manual		PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600
		TASK NO. F95600	WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Control Data Corporation: Althoff, J. L., Apiceila, M. L.			
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4 / 1 / 87 - 12 / 31 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30	15. PAGE COUNT 49
16. SUPPLEMENTARY NOTES WRDC/MTI Project Priority 6203			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)	
FIELD	GROUP	SUB GR.	
1308	0905		
19. ABSTRACT (Continue on reverse if necessary and identify block number)			
<p>This manual explains to application programmer's how to use the Neutral Data Manipulation Language (NDML). It also explains the syntax and semantics of each NDML command.</p> <p>BLOCK 11:</p> <p>INTEGRATED INFORMATION SUPPORT SYSTEM Vol V - Common Data Model Subsystem</p> <p>Part 8 - Neutral Data Manipulation Language (NDML) Reference</p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson		22b. TELEPHONE NO. (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC/MTI

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<u>SUBCONTRACTOR</u>	<u>ROLE</u>
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

		<u>Page</u>
SECTION	1.0 INTRODUCTION	1-1
SECTION	2.0 SYSTEM OVERVIEW	2-1
SECTION	3.0 NDML COMMANDS	3-1
	3.1 Data Retrieval Commands	3-2
	3.2 SELECTION COMBINATION Command	3-18
	3.3 DELETE Command	3-19
	3.4 INSERT Command	3-22
	3.5 MODIFY Command	3-27
	3.6 Transaction Commands	3-30
	3.6.1 BEGIN TRANSACTION Command	3-30
	3.6.2 UNDO and ROLLBACK Commands	3-30
	3.6.3 COMMIT Command	3-30
	3.7 Loop Construct	3-31
	3.7.1 When a Loop Construct Is Needed	3-31
	3.7.2 Syntax	3-31
	3.7.3 NDML Loop Control Statements	3-32
	3.7.4 Evaluation	3-32
	3.8 Distributed Update Restrictions	3-34
	3.9 Error Codes	3-36
SECTION	4.0 NDML PRECOMPILER OVERVIEW.....	4-1
APPENDIX	A BNF OF THE NDML	A-1
	A.1 Conventions	A-1
	A.2 NDML Backus-Normal Form (BNF)	A-2

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SECTION 1

INTRODUCTION

The Neutral Data Manipulation Language, hereafter NDML, was developed by the U. S. Air Force to provide access to the databases of the Integrated Information Support System (IISS) Testbed. The NDML allowed its users to work with the heterogeneous, distributed databases on the IISS Testbed as though they constituted a single relational database.

The NDML has been designed to provide as much functionality as possible while attempting to be logical in application and convenient. The NDML is intended to be used by data processing personnel and by manufacturing personnel who may have limited knowledge of database systems.

The NDML is a language similar to SQL (Structured Query Language, pronounced "sequel") and Quel, two well-known languages used to access relational databases. The utility of the access method provided by the NDML is supported by extensive theory and practical tests of these relational languages.

The NDML is designed for use either as a stand-alone language or as embedded statements in the host languages of COBOL or FORTRAN. Currently, only embedded statements are supported and this manual applies only to embedded NDML. The NDML examples in the command descriptions in this manual neglect the embedding characters (*# for COBOL or C# for FORTRAN) for simplicity, but their use is shown in succeeding sections.

When stand-alone requests are supported, deviations from the embedded language will be as few as possible. The differences are due mainly to the requirement that a retrieved table be presented to host programs a row at a time, while the entire table can be presented in response to a stand-alone request from an interactive NDML user.

The NDML Precompiler is used to process the application program containing embedded NDML statements before the host language (COBOL or FORTRAN) compiler is used. The host language compiler can be used first to debug host language statements, but the NDML precompile step must precede host-language compilation before executable object code is produced. The use of the Precompiler is described in the NDML Precompiler Users Manual.

The important property of the NDML to keep in mind when using this manual is that you perceive all data to be in the form of tables. Data within the database can be considered to be stored as tables even if containing only one row of values. Similarly, only tables can be retrieved from the database, even if the table consists of a single "row" with a single "column" (i.e., only a single value). This important property of relational databases allows the output of one retrieval command to be utilized as the input to another operation without worrying about the structure of the data. Furthermore, "chunks" of data can be retrieved and used without having to specify the structure of the data for each application and the size of the data chunk.

Tables are usually called "relations" and the terms table and relation will be used synonymously here. Similarly, rows of the table may be called "records" or "tuples" and columns may be called "data fields", "data items" or "attributes". An individual number or character string entry in the table will be called a "value".

Each of the following sections on specific commands begins with the syntax of the command. The syntax is presented using a method that is described at the beginning of Section 3; it is similar to the method used in the NDDL manual. The rigorous BNF description of the language is presented in appendix A. Following the syntax of the command, semantic notes point out conflicting commands and restrictions that are not supported by the system. This document should provide sufficient information for an application programmer to begin work. Moreover, you will find it an appropriate reference in the future when you have become familiar with NDML.

If you are unfamiliar with SQL, you should consult tutorials and references on that language before using this guide.

References include:

Chamberlin, D.D., et al., "Sequel 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of Research and Development. Vol 20, No. 6, Nov. 1976, pp. 560-575.

Date, C.J., A Guide to DB2. Addison-Wesley Publ. Co., 1984.

In addition, many commercial relational database systems offer interface languages similar to SQL. The manuals for these languages are useful for becoming acquainted with the general syntax of SQL.

SECTION 2

SYSTEM OVERVIEW

The processing system is known as the Common Data Model Processor (CDMP). The CDMP provides the application programmer with important capabilities to:

- Request database accesses in a non-procedural data manipulation language (the NDML) that is independent of the data manipulation language (DML) of any particular data base management system.
- Request database access using a NDML that specifies accesses to a set of related records, rather than to individual records (i.e., using a relational DML).
- Request access to data that are distributed across multiple databases with a single NDML command, with minimum knowledge of data locations or distribution details.

Information about external schemas, the conceptual schema and internal schemas (including data locations) are provided by CDMP access to the Common Data Model (CDM) database. The CDM is a relational database of metadata pertaining to IISS. It is described by the CDM1 information model using IDEF1. The Precompiler parses the application program source code, identifying NDML commands. It applies external-schema to conceptual-schema and conceptual-schema to internal-schema transforms on the NDML command, thereby decomposing the NDML command into internal-schema, single database requests. These single database requests are each transformed into generic data manipulation language (DML) commands. Programs are generated from the generic DML commands which can access the specific databases to accomplish the request. These programs, referred to as Request Processors (RP), are stored at the appropriate host machines. The NDML commands in the application source program are replaced by host-language code which, when executed, activates the run-time request evaluation processes associated with the particular NDML command.

The Precompiler also generates a CS/ES Transformer program which will take the final results of the request, stored in a file as a table with external-schema structure, and convert the data values into the correct form for presentation. The CS/ES Transformer also performs NDML function operations on the data.

Finally, the Precompiler generates a Join Query Graph and Result Field Table which are used by the Distributed Request Supervisor (DRS) during the run-time evaluation of the NDML request.

The DRS is responsible for coordination of the run-time activity associated with the evaluation of an NDML command. It is activated by the application program, which sends it the names and locations of the query processors to activate along with run-time parameters which are to be sent to them. The results generated by the query processors are stored as files in the form of conceptual-schema relations on the host which executed the query process. Using the Join Query Graph, transmission cost information and data about intermediate results, the DRS determines the optimal strategy for combining the intermediate results of the NDML command. It issues the appropriate file transfer request, activates aggregators to perform unions, joins, and NOT IN SET operations, and activates the appropriate CS/ES Transformer program to transform the final results. Finally, the DRS notifies the application program that the request is completed, and sends it the name of the file which contains the results of the request.

The Aggregator is activated by the DRS. An instance of the Aggregator is executed for each union, join, and NOT IN SET operation performed. It is passed information describing the operation to be performed and the file names containing the operands of the operation. The DRS ensures that these files already exist on the host which is executing the particular Aggregator program. The Aggregator performs the requested operation and stores the results in a file whose name was specified by the DRS.

SECTION 3

NDML COMMANDS

The following conventions are used in the description of the NDML commands at the beginning of the following subsections:

3.1 Conventions

3.1.1 Notation

UPPER CASE WORDS denote keywords in the command

LOWER CASE WORDS denote user-defined words (entered in upper case)

{ } denotes that exactly one of the options within the braces must be selected by the user

• • • denotes repetition of the last element

[] denotes that the entry within the brackets is optional

| denotes an "or" relationship among the entries

_ denotes default option

3.1.2 Punctuation

1. A "." is used to separate the table-label (i.e., table alias) from the column-name. The table-label is used to match a column to a specific table in the list of tables referenced in the FROM clause.
2. A ":" is placed before the name of a host-language program variable, structure or file name that will receive returned values.
3. A "," is inserted between entries in the list of tables in a FROM clause.
4. A "," is inserted between subscripts to an array variable.
5. Parentheses are used to enclose the column-list in an INSERT statement.
6. Parentheses are used to enclose the object column of a function.
7. Parentheses are used to enclose the values to be inserted in an INSERT statement.

8. Parentheses are used to enclose a program variable subscript list.
9. A mandatory ";" or "loop construct" (see Loop Construct, subsection 3.7) is affixed at the end of the command.

3.1.3 Character Case

Only upper-case letters are recognized by the NDML Precompiler.

3.1.4 Word Length

Table labels are limited to 2 characters.

Table and column names are defined by the relational view in use.

3.2 Data Retrieval Commands

3.2.1 Syntax

Data are retrieved from the database using the SELECT command. The command has the following syntax:

```
SELECT [WITH { EXCLUSIVE } LOCK]
        { SHARED  }
        { NO      }

        [INTO      { FILE 'file-name'      } ]
              { FILE ':variable-name'    }
              { STRUCTURE :variable-name }

        [DISTINCT] { [table-label] ALL
                     { :variable-name [(subscript, ...)] = expr-spec ... }
        FROM table-name [table-label], ...
        [WHERE predicate-spec
        [AND predicate-spec ...]]
        [ORDER BY column-spec [direction] ...]
              { ;
              { loop construct }
```

where

file-name and variable-name are defined in the host program,

table-label is a one- or two-character name,

table-name and column-spec are defined for the relational view,

predicate-spec is either a column-, join-, between-, or null-predicate (see Appendix A)

subscript is an integer or a subscript-list

direction is {
ASC
DESC
ASCENDING
DESCENDING
UP
DOWN

expr-spec is { column-spec ([DISTINCT] column-spec)
{
AVG
MEAN
MAX
MIN
SUM
COUNT

column-spec is { column-name }
{ table-name.column-name }
{ table-label.column-name }

loop construct is a list of program and/or NDML statements enclosed in braces for the purpose of transferring retrieved values to program variables, processing host language statements with the values retrieved, etc.

3.2.2 Comments

(a) SELECT Keyword

The SELECT command is the only command used in NDML to retrieve data from the distributed database. This keyword must be the first word in the command.

(b) LOCK Phrase

A lock limits access to specific rows of tables while a transaction is being processed to prevent alteration these rows during the transaction. A lock is owned by the transaction in which the SELECT statement occurs. An EXCLUSIVE lock denies access to all rows accessed by the transaction to all other processes. In addition, a request by any other transaction for any type of lock on the row will be caused to wait until the EXCLUSIVE lock is released. An EXCLUSIVE lock is normally used only when using an update command on a row, but might be needed in a SELECT request in a transaction to ensure that no other transaction can obtain a lock on the row. A transaction issuing a SELECT request may need to lock a selected row if it intends to update the row based on values retrieved earlier.

A SHARED lock also locks rows but allows other transactions also to lock a row. A SHARED lock is used normally in a SELECT command to ensure that a row is not changed by a contemporary MODIFY or DELETE transaction that must obtain an EXCLUSIVE lock to perform its function.

If no type of lock is specified in a lock-request, a NO lock is assumed unless the SELECT falls within an explicitly specified transaction. For example,

BEGIN TRANS

COMMIT/UNDO;

causes a SHARED lock to be requested automatically.

The lock placed by a transaction depends on the implementation of locks in the particular database systems of the internal schema. The lock placed on the data in the internal schema by the local database manager usually locks either (1) only the accessed record or (2) the entire accessed table, depending on the local database. A "LOCK TABLE" command that will ensure that an entire table, rather than just a record, is locked is not provided in NDML at present. You should assume only that each record accessed is locked.

(c) INTO Phrase and Variable Assignments

The data retrieved by a SELECT command can be either (1) placed into a file or program structure with the INTO phrase or (2) assigned to program variables using a variable-assignment construct. Selecting into a program structure is not applicable if you embedded the NDML statement in a FORTRAN program because structures do not exist in the language.

The file name can be specified by using the keyword FILE and enclosing the file name in single or double quotes. If a colon is not the first character following the first quote, then the literal contents of the quoted character string will be taken to be the name of the file. If the first character following the first quote is a colon, then the rest of the character string will be taken to be the name of a program variable, the contents of which is the name of the file.

There is no default extension or file type for the filename specified in the SELECT statement. You must explicitly state the extension or file type as part of the file name.

You must supply either the COBOL SELECT and FD layout or the FORTRAN format statement for the file if the file is to be accessed by the application program.

The file specified on the SELECT statement does not have to have been previously created. Code will be generated into the application program to create, open, populate, and close the file. Note, that during execution of the actual modified application program, the file will be closed when you receive control after the completion of the NDML processing.

The entire result of the SELECT will be placed in the file, one row per record, in the order normally produced by the SELECT command. A loop construct should not be specified when the INTO phrase is used to place the results in a file. There will be a one character (COBOL PIC 9 or FORTRAN CHARACTER *1) null value flag at the beginning of the record for each column selected; 1 means null, 0 means null. The null field itself will contain the null value specified in the NDDL DEFINE DATABASE command.

If the user has embedded the NDML statement in a FORTRAN program and is selecting into a file, the following rules apply. If selecting a character data item, the exact size of the data item will be allotted in the record. If selecting the statistical function COUNT, 9 spaces will be allotted. If selecting a floating point data item, or the statistical functions AVG, SUM, or MEAN, 19 spaces will be allotted in the record. These 19 spaces will be in character format. When the user accesses the final result file, a conversion routine must be called to convert the 19 character string to a floating point value. If selecting an integer data item, 10 spaces will be allotted in the record. These 10 spaces will also be in character format. As with the floating point, a conversion routine must be called to convert the 10 character string to an integer value. If selecting the statistical functions MIN or MAX, the size allotted will depend on the type of the operand. Character results will be left justified in the space allotted for the result. Numeric results will be right justified in the space allotted for the result.

A structure is indicated to receive the retrieved data by the keyword STRUCTURE followed by a space, a colon and the program name of the structure. The defined data types for the fields in the structure must agree exactly with those for the corresponding column. For a structure target, only the first row returned will be placed in the target unless the application program contains code for a loop construct following the SELECT command. The syntax of the loop construct is described in subsection 7.2.

If the user has embedded the NDML statement in a FORTRAN program and is selecting into program variables, the following rules apply. If selecting a floating point data item, the result variable must be defined as DOUBLE PRECISION. If the data item is integer, the result variable must be defined as INTEGER. If the data item is character, the result variable must be defined as CHARACTER *n, where n is the value of the external schema size. If statistical functions MEAN, AVG, or SUM are used, the result variables must be defined as DOUBLE PRECISION. If statistical function COUNT is used, the result variable must be defined as INTEGER. If statistical function MIN or MAX is used, the result variable must be defined according to the data type of the data item being selected.

If neither a file, a structure, nor variables are specified to receive the result of the select command in embedded NDML within an application program, the Precompiler will reject the NDML SELECT statement. Thus, an assignment of retrieved columns to program variables or an INTO clause must be specified, but both cannot be specified. Also note that if ALL is specified for columns, an INTO phrase must be specified.

The following are examples of valid SELECT statements.

```
SELECT INTO FILE 'DEPT-FILE'
      D.DNO D.DNAME D.DLOC D.DSIZE
FROM DEPT D
ORDER BY D.DNO;
SELECT INTO STRUCTURE :DEPT-STRUCT
      D.DNO D.DNAME D.DLOC D.DSIZE
FROM DEPT D
ORDER BY D.DNO
loop construct
SELECT :DEPTNO = D.DNO :DEPTNAME = D.DNAME
      :DEPTLOC = D.DLOC :DEPTSIZE = D.DSIZE
FROM DEPT D
WHERE D.DLOC != 'LAX'
loop construct
```

(d) SELECT DISTINCT Phrase

The DISTINCT clause on a SELECT statement is used to specify that duplicate rows are to be removed prior to presentation of the results. Omitting the DISTINCT clause implies that duplicate rows are not removed unless you specified this clause at NDDL Create View time. If DISTINCT clause is specified in both the NDML select and the NDDL Create View, the result is the same as if it had been specified.

The DISTINCT phrase refers to the entire set of selected columns following it. For example, SELECT DISTINCT ALL FROM T1 removes only those rows from T1 for which all column values are identical to those of another row in T1. The DISTINCT processing is applied to rows in their external-schema formats.

```
SELECT INTO FILE 'FILE-NAME' DISTINCT ALL
      FROM DEPT D
      WHERE D.LOC = 'LAX';
SELECT INTO FILE 'FILE-NAME' DISTINCT
      D.DNO D.DNAME D.LOC
      FROM DEPT D
      WHERE D.SIZE = 'LARGE';
```

(e) Restrictions on Column Specifications

Only columns from a table can be specified; quoted literal data to be duplicated in a column are not allowed, but can be introduced easily by the application programmer. Arithmetic expressions involving column data are also not supported; they also can be implemented easily directly in the application program. For example, the following commands are not supported:

```
SELECT INTO FILE 'FILE-NAME'
      EMP ' IS IN DEPARTMENT ' EMPDEPT
      FROM EMP;
SELECT INTO FILE 'FILE-NAME'
      'OVERHEAD IS ' 0.5 * AMOUNT
      FROM CONTRACTS;
```

The column specification ALL indicates all columns of the single table specified by the rest of the SELECT statement. The table can be derived from a single table indicated in the FROM clause, as (optionally) qualified by a WHERE clause. Alternatively, multiple tables can be specified in the FROM clause if a join operation is specified in a WHERE clause, but columns from only one table can be retrieved at a time using the ALL specification. For example, the following query is not supported:

```
SELECT INTO FILE 'FILE-NAME' ALL 1
      FROM TABLE1, TABLE2;
```

but these following queries are supported:

```
SELECT INTO FILE 'FILE-NAME' ALL
      FROM TABLE1, TABLE2
      WHERE TABLE1.CITY = TABLE2.CITY;
SELECT INTO FILE 'FILE-NAME' E.ALL
      FROM EMP E, DEPT D
      WHERE E.DNO = D.DNO;
```


An important requirement that must be observed to use the ALL column specification is that an INTO phrase must indicate where to place the results of the SELECT because individual columns cannot be explicitly assigned to program variables in this syntax. The number of data fields and data types in the target structure or file must correspond to those of the columns, as discussed in the subsection above on the INTO phrase. In the ALL specification, the layout of the retrieved columns is in alphabetical order. For example, if there are columns TEAM_NO and TEAM_NAME, their order of retrieval will be TEAM_NAME first and then TEAM_NO. However, the ALL specification is prone to error in embedded NDML because the number and order of the columns can change if the table is reorganized. Note also that the ALL specification can refer to only one table. If more than one table is specified in the FROM clause, the appropriate table to which the ALL designation applies must be indicated using a table-label.

(f) Statistics Functions

Function expressions can be presented as the result of a SELECT statement only; they cannot be used in a WHERE or ORDER BY clause. These functions are used to specify that column statistics of AVG value, MAX value, MIN value, SUM value, or COUNT of rows are to be produced. AVG and MEAN are synonyms.

The results of AVG (column) are the same as the results of SUM(column)/COUNT(column). All values are considered unless the optional DISTINCT phrase within the function clause is included; in which case, duplicate values are removed prior to the function application. Null values do not contribute to the SUM, MAX, MIN or COUNT function.

SELECT cannot return both a table and the result of functions in a single statement. Thus, if one function is specified in an expr-spec, then all values to be retrieved must be the one result of functions. It is permissible to retrieve the several functions, but the user should be aware that the values in the single row returned will not necessarily have any logical relationship.

MIN, MAX and COUNT can be applied to both numeric and string columns. AVG, MEAN and SUM can be applied only to numeric columns. Functions are applied to columns in their external-schema formats. Statistic functions ignore nulls in the data. For the empty set, COUNT returns zero and other functions return an undefined result; the existence of the empty set for non-COUNT functions results in a condition code set in NDML-STATUS, as discussed below.

The ORDER BY clause should not be used when functions are specified because unnecessary processing will be performed (the system may not allow the clause to be specified). Specification of function DISTINCT before MIN or MAX is ignored. Functions cannot be used in a WHERE clause because the result of a function is a property of a group of rows rather than of each row. A SELECT DISTINCT specification should not be used with functions because it causes unnecessary processing.

The formats of function results in COBOL are AVG, MEAN and SUM: S9(9)V9(9); COUNT: S9(9). The formats of function results in FORTRAN are AVG, MEAN and SUM: F19.9; COUNT: 19. The number of rows returned by the request is contained in the variable NDML-COUNT (or NDMLCT in FORTRAN) and generated into the application program by the Precompiler; obviously, it will always have a value of one for function requests. The variable NDML-STATUS (or NSTATS in FORTRAN) is generated into the applications program by the Precompiler; obviously, it will always have a value of one for function requests. The variable NDML-STATUS (for NDMLST in FORTRAN) generated into the application program contains a code that indicates the success or failure of the request. An all zero code indicates successful completion; any other code indicates an error. If a function operates on an empty column, a result may be returned that is not really valid (for example, SUM will return 0.). The NDML-STATUS (or NDMLST in FORTRAN) flag and associated null indicator should be checked by the application program before using the result returned by a function.

```
SELECT INTO FILE 'FILE-NAME'
    AVG(P.LEAD-TIME) MIN(P.LEAD-TIME) MAX(P.LEAD-TIME)
    FROM PART P WHERE P.SIZE > 100;
SELECT INTO FILE 'FILE-NAME' COUNT(D.DNAME)
    FROM DEPT D, EMP E
    WHERE E.DNO = D.DNO;
SELECT INTO 'FILENAME.DAT'
    MIN(SE.SALARY) MIN(HE.RATE)
    FROM SALARIED-EMP SE, HOURLY-EMP HE;
SELECT INTO FILE 'FILE-NAME' COUNT(DISTINCT E.JOB)
    FROM EMP E
    WHERE E.DNO = 10;
SELECT INTO FILE 'FILE-NAME' COUNT(DISTINCT D.LOC)
    FROM DEPT D, NEWDEPT N
    WHERE D.NAME = N.DNAME;
```

Note: User-defined functions and explicit arithmetic functions (e.g., WEIGHT * 2.2) are not supported in this release.

(g) FROM Clause

Table labels or table names may or may not be required by the syntax of the particular request. If two or more tables are specified in the table-list, it is a good idea to be concise and use table labels or table names to designate columns. When a table is joined with itself, it is necessary to use table labels to distinguish columns.

(h) WHERE Clause

The WHERE clause is used to limit the information returned from one or more tables. If the WHERE clause is not specified, all rows from the first table indicated in the table-list are returned.

Only column-predicate or join-predicate comparisons are allowed in WHERE clauses. The column-predicate compares the value of a column with a single specific value indicated by the contents of a scalar program variable, a literal string in quotes, or a number. Either the column name or value can be the first object of the comparison (only the case in which the value is second is shown in the syntax above). AND clauses can be used to specify multiple qualifications on the table selected. The comparison operator (bool-op) includes most common operations but does not include an "IN" comparison that would allow a column to be compared with many values. If there are any join-predicate comparisons in the WHERE clause, they must all be listed first or all listed last. They cannot be interspersed with the column-predicate comparisons.

The qualifications specified in the WHERE clause of an NDML statement will be "ANDed" with those specified in the WHERE clause of the CREATE VIEW. These qualifications include the entity to entity joins (equi-joins and outer-joins) and the column to value qualifications (including parenthesis, NOT, AND, OR, XOR, Boolean operators, Between and Null operators). The precompiler will ignore the WHERE ALL qualifications of the NDML statement when CREATE VIEW WHERE clause will be enforced by the NDML precompiler. A description of the CREATE VIEW command may be found in the Neutral Data Definition Language (NDDL) User's Guide.

The NDML command can easily be placed within a user-defined program loop within an application program. Consequently, subqueries, in which the comparison values are returned by another SELECT request, are not supported because more than one value can be returned by the subquery. Other possible comparison operators currently not supported include EXISTS, ALL and ANY.

Note that changing the contents of a program variable within the loop construct of the SELECT command will have no effect on the result because the query has already been executed before the loop construct is activated. A loop construct is used only to transfer data from a completed SELECT query to program variables or to a structure. The loop construct is described in subsection 3.9.

Supported Query:

```
SELECT INTO FILE 'FILE-NAME' DNO DNAME
FROM EMP
WHERE DTYPE = 'SALES'
AND DLOC = 'SOUTH';
```

Unsupported Query:

```
SELECT INTO FILE 'FILE-NAME' DNO DNAME
FROM DEPT
WHERE DNO IN
  (SELECT DNO
   FROM LOCATION
   WHERE DEPTLOC = 'LA');
```

The join-predicate comparison allows only the equi-join (=) and NOT IN SET (!=) operations; the operators <, <=, > and >= are not implemented. The join fields compared in a join or NOT IN SET operation need not have identical data types in the user's (external) view of the table, except that numeric data must be compared with numeric data and character strings with character strings.

The equi-join connects a row from each of two tables to form one row in the result table if the values in the specified columns in the tables are identical. Duplicate rows will be returned if duplicate rows exist in either table. Rows for which a match is not found are not included in the result table.

The outer-join operation is a selection procedure that is similar to join, but when rows from the table specified to the left of the != operator do not match entries in the table to the right, a "partial" results row will be considered for retrieval. For a more detailed explanation, refer to C.J. Date, *An Introduction to Database Systems*, 3rd Edition. Because columns from the table on the right may have been selected, null values may be introduced. The query may specifically exclude or include those "partial" rows by use of the IS[NOT] NULL predicate applied to one of the columns from the right table. For example, with the following request:

```
SELECT INTO 'FILENAME.TMP' D.DNO D.DNAME E.NAME
FROM DEPT D, EMP E
WHERE D.DNO != E.DNO;
```

if the following data are found,

<u>D.DNO</u>	<u>E.DNO</u>
1	2
2	3
4	5
5	6
7	8
8	9

the result will have "full" rows (without a null E.ENAME) for

<u>D.DNO</u>
2
5
8

and "partial" rows (null E.NAME) for

E.DNO
1
4
7

since these departments did not have employees.

In the example below, the outer-join is applied to the employee table. This is useful often in validating the subset constraints (entity dependence) defined in the conceptual schema.

```
SELECT INTO 'FILENAME.FOR' E.DNO E.ENAME  
      D.DNO D.DNAME  
FORM DEPT D , EMP E  
WHERE E.DNO U = D.DNO ;
```

The result will have "full" rows for

E.DNO
1
4
7

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The CDM Administrator (CDMA) should inform the user of these restrictions before precompilation. The CDMA can determine those by examining conceptual-internal schema mapping relationships.

(i) ORDER BY Clause

The ORDER by clause is used to specify the sequencing rules for presentation of the results of a SELECT operation. Omitting the ORDER by clause on a SELECT statement implies that the rows of the result table are presented in a system-determined order.

The columns in the order-spec-list control the sorting of result rows in major-to-minor order. If the direction phrase is omitted for a column, then ASC (ascending) is assumed. The columns of an order-spec-list need not all have the same accompanying direction. Also, the columns need not appear in the column-list of the SELECT phrase.

Sorting is done on the columns in their external-schema formats and will be done on the machine running the application program. The order of the sorted result will depend on the storage code used by the computer running the applications program. ASCII is to be used whenever possible. Thus, the result of the same program can differ if it is precompiled and run on different machines. Note that ASC, ASCENDING and UP are equivalent and that DESC, DESCENDING and DOWN are equivalent. Null values are treated as the largest representation and will appear last when the ASCENDING option is chosen and first when the DESCENDING option is chosen.

```
SELECT INTO 'FILENAME.COB' E.NAME E.DEPT E.PHONE
      FROM EMP E
      WHERE E.JOB CODE > 50
      ORDER BY E.NAME;
SELECT INTO FILE 'FILE-NAME' PART# SIZE
      FROM PART
      ORDER BY SIZE DESCENDING;
SELECT INTO 'FILENAME.DAT' D.DEPT# D.LOC D.CITY
      FROM DEPT D
      ORDER BY D.CITY ASC D.LOC DESC D.SIZE ASC;
```

(j) Nulls

Nulls are intentionally introduced into a query result by use of the outer join along with selection of columns from the second table when the data does not match. These can be tested for with the IS [NOT] NULL predicate. A value to be recognized as NULL for the internal schema databases will be stored in the CDM. These shall be used when qualifying (use of IS [NOT] NULL) on an external schema data item that maps to this internal schema value, outside of the outer join application.

To allow testing of retrieved data values within the loop construct a COBOL condition (or FORTRAN string) will be added to the user's program that can be tested with an IF statement. This condition actually will be an array of flags, one occurrence for each selected item, left to right. The COBOL or FORTRAN element (FLAG-X in COBOL and FLAGAR in FORTRAN) is set to 1 if a null value is encountered, zero otherwise. If a null value is found, the value of the user's variable will be set to the null value specified in the NDDL DEFINE DATABASE command. For example,

```
SELECT :X1 = A.B :X2 = A.C :X3 =A.F :X5 = A.K
FROM      TABL1      A
{
IF FLAG-X (3) =1
}
```

In this example, the user has tested the column A.F for a potential null value. The user must be careful with nested selects in one routine. The null-column array only refers to the last row returned (the inner most SELECT on nested selects).

```
SELECT
{
test here for outer SELECT NULLS
SELECT
{
test here for inner SELECT NULLS
}
test here for inner SELECT NULLS of last inner row retrieved
}
```

As can be seen, the null column array is not a "stack" of flags.

In FORTRAN, the array "FLAGAR" can be tested for a non-zero value:

```
SELECT :X1 = A.B :X2 = A.C :X3 = A.F :X5 = A.K

FROM      TABL1      A
{
IF (FLAGAR(3:3)      .NE. '0')      THEN
      ENDIF
}
```

(k) Grouping Clauses

This release does not support GROUP BY and HAVING clauses to determine aggregate properties of multiple rows of a table. These operations must be performed by the application process.

(l) Logic Rules for WHERE clause

NOT

The NOT operator will be translated according to DeMorgan's Law: Operators are reversed, AND becomes OR and OR becomes AND.

EXCLUSIVE OR

(See Appendix A for exact syntax definitions)

```
WHERE      X.A < 5 XDR X.B = 12
```

will be translated to

WHERE (X.A < 5 OR X.B = 12) AND (X.A >= 5 OR X.B != 12)

WHERE NOT (X.A < 5 XOR X.B = 12)

will be translated to

WHERE (X.A > 5 AND X.B != 12) OR

(X.A < 5 AND X.B = 12)

The XOR translation is done before the NOT translation.

BETWEEN

A column can be compared to other columns of the same table, literal values, numeric constants, or program variables. The statement translates as follows:

WHERE A.X BETWEEN 7 AND :VAR-X

will be translated to

WHERE (A.X >= 7 AND A.X <= :VAR-X)

and

A.X NOT BETWEEN 'AAA' AND 'KKK'

will be translated to

(A.X < 'AAA' OR A.X > 'KKK').

BETWEEN is understood to be inclusive of the end points.

NOTEQUAL

Note that both <> and != are allowed for inequality tests.

(m) Logic Definitions

These definitions are adapted from the current draft of the proposed SQL standard.

In a single column-predicate:

1. Let X denote the result of the first value or column-spec and let y denote the result of the second value or column-spec. The values x and y must be comparable values.
2. $(x \text{ bool-op } y)$ is unknown if x and y must be comparable values.
3. If x and y are non-null values, $(x \text{ bool-op } y)$ is either true or false:
 - $(x = y)$ is true if x and y are equal.
 - $(x \neq y)$ is true if x and y are not equal.
 - $(x < y)$ is true if x is less than y .
 - $(x > y)$ is true if x is greater than y .
 - $(x \leq y)$ is true if x is not greater than y .
 - $(x \geq y)$ is true if x is not less than y .
 - $(x \neq y)$ is true if x and y are not equal.
4. Numbers are compared with respect to their algebraic value.
5. The comparison of two character strings is determined by the comparison of $\langle \text{characters} \rangle$ with the same ordinality. If the character strings do not have the same length, the comparison is made with a temporary copy of the shorter character string which has been extended on the right with space characters so that it has the same length as the other character strings.
6. Two character strings are equal if all $\langle \text{characters} \rangle$ with same ordinality are equal. If two character strings are not equal, their relation is determined by the comparison of the first pair of unequal $\langle \text{characters} \rangle$ from the left end of the character strings. This comparison is made with respect to implementor-defined collating sequence.
7. Although $(x = y)$ is unknown if both x and y are null values, a null value is identical to or is a duplicate of another null value.
8. $(x \text{ IS NULL})$ is either true or false.

9. (x IS NULL) is true if x is the null value.
10. (x IS NOT NULL) has the same result as NOT(x IS NULL).
11. NOT(true) is false, NOT(false) is true, and NOT(unknown) is unknown. AND and OR are defined by the following truth tables:

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

12. As implied by the syntax, expressions within parentheses are evaluated first and when the order of evaluation is not specified by parenthesis, NOT is applied before AND, AND is applied before OR, and operators having the same precedence level are applied from left to right.

(n) Mapping Rules for Select, Query-Combination

The NDML precompiler will be modified to choose a secondary copy of data for retrieval if the requesting process is on the same host and if the CDMA has permitted it through the use of the host and if the CDMA has permitted it through the use of the ALLOW RETRIEVAL clause of the CREATE MAP command. If DIALLOW RETRIEVAL has been specified, which is also the default, only the primary copy of data will be retrieved. A description of the CREATE MAP command may be found in the Neutral Data Definition Language (NDDL) User's Guide.

(o) Mapping Rules for Precompiler Generated Referential Integrity Checks

The NDML precompiler will continue to select the primary copy of data.

3.3 SELECTION COMBINATION Command

3.3.1 Syntax

SELECT

```
[INTO      { 'file-name'      }]
           { ':variable-name'  }
           { STRUCTURE :variable-name }
```

[DISTINCT]

:variable-name [(subscript,...)] ...

FROM

simple-select-combination

```
{ ;          }
{ Loop construct }
```

where simple-select-combination is a parenthesized combination of simple selections using the operators UNION, DIFFERENCE and INTERSECT.

A simple select is

```
SELECT      [DISTINCT] column-list
FROM        table-list
[WHERE      predicate-list].
```

A more precise specification of allowable syntax is contained in the appendix, as rule named query-spec.

The following rules have been derived or extracted from the draft ANSI proposed SQL standard of December, 1985.

1. The simple, or inner selections within parentheses are evaluated first, and when the order of evaluation is not specified by parentheses, INTERSECT is applied before UNION or DIFFERENCE and the set-operators at the same precedence level are applied from left to right.
2. Each inner selection is evaluated and stored in temporary tables.
3. Let T and T' denote tables. The result of T set-operator T' is a table R, derived as follows:

Case:

4. If the set-operator is UNION, then:
 - a. Initialize R to an empty table.
 - b. Insert each row of T and each row T' into R.
5. If the set-operator is INTERSECT, then:
 - a. Initialize R to an empty table.
 - b. For each row of T, if a duplicate of that row exists in T', insert that row of T into R.
6. If the set-operator is DIFFERENCE, then:
 - a. Initialize R to an empty table.
 - b. Insert each row of T into R.
 - c. For each row of R, if a duplicate of that row exists in T', eliminate that row from R.
7. T and T' must have the same number of columns. Corresponding columns in T and T' must have identical data type descriptions.
8. The degree of R is the same as the degree of T and T'.
9. The columns of the final relation representing the combinations of the inner selections are mapped by the first selection (before the FROM keyword). The columns of the final result can be mapped into a file, program variables or an internal structure. DISTINCT operations can be applied to the final resultant table and to the inner selection.

3.4 DELETE Command

3.4.1 Syntax

The DELETE Command removes rows from an external-schema table. The DELETE command has the following:

```
DELETE FROM table-name [table-label]
      [USING table-name [table-label], ...
      WHERE { ALL
            { predicate-spec
```

where

table-label is a one-or two-character name,

table-name and column-name is defined for the relational view,

predicate-spec is either a column-, join-, between-, or null-predicate (see Appendix A).

3.4.2 Comments

(a) Locking

A DELETE command inside a transaction usually places a "key lock" on deleted rows until a COMMIT command is encountered. This lock ensures that another process cannot insert a row with the key of the deleted row until the DELETE action has been finalized by a COMMIT command. A DELETE command outside of a transaction is usually committed immediately and no lock is used. Actual lock mechanisms depend on the internal-schema databases.

(b) USING Clause

The USING clause specifies tables that are accessed by the WHERE clause to qualify the request. These tables are not to have rows deleted from them. To be meaningful, tables indicated in the USING clause must be related to the table on which the DELETE command acts by a join-predicate.

(c) WHERE Clause

The WHERE clause is used to specify which rows qualify to be deleted. The WHERE clause is mandatory and the Precompiler will reject the request if it is not present. If all the rows of a table are to be deleted, the WHERE ALL clause should be used. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement. Because distributed update is not supported, a WHERE clause mapping to multiple subtransactions per preference (horizontal partitioning) is not supported, since the query results of one subtransaction would dictate the actual rows to be deleted by another subtransaction.

The qualifications specified in the WHERE clause of an NDML statement will be "ANDed" with those specified in the WHERE clause of the CREATE view. These qualifications include the column to value predicates and may be expressed within nested parentheses. This parenthesized logic will be enforced by the precompiler. The NDML precompiler will ignore the WHERE ALL qualification of the NDML statement when CREATE VIEW qualifications exist.

Note: It is permissible to modify rows in the view that are moved thereby out of the view.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The CDM Administrator should inform the user of these restrictions before precompilation. The CDMA can determine these by examining the conceptual-internal schema mapping relationship.

(d) Mapping Restrictions

The external-schema table (your view of the table) must map to one complete conceptual-schema entity class. This means that a request to DELETE a row in a table in your view can be rejected by the system because other information that you are not (necessarily) aware of would also have to be deleted in the conceptual-schema representation of the database. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct DELETE command to explicitly delete all the columns of a row in the conceptual schema.

The entity class (in the conceptual schema) may map to just part (or all) of one or more record types in the actual database (in the internal schema). If just part of a record type is mapped to, that deleted part is filled with null-values and the remainder is left as is. The null values used are those specified in the CDM.

The NDML precompiler will generate update transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be updated. The update of these secondary copies are not guaranteed.

Furthermore, the precompiler will not reject multiple subtransactions being generated by an update action, one subtransaction per copy of data. The precompiler will continue to reject cases, where for a specified preference one entity class maps to a non-normalized database structure resulting in multiple subtransactions.

(e) Integrity Constraints

A request to delete a conceptual-schema entity that has dependent entities will be rejected at runtime. Those dependent entities cannot be ignored; their existence depends on the existence of the independent entity.

A future release may support DELETE WITH CASCADE, which will delete any dependent entities associated with the specified entity.

(f) Null Values

The specification of internal-schema null-values is DBMS dependent. The values specified in the CDM will be stored when required.

(g) Examples:

```
DELETE FROM OFFER F
  WHERE F.STATUS = 'EXPIRED';
```

```
DELETE FROM OFFER
  WHERE ALL;
```

```
DELETE FROM OFFER F
  WHERE F.STATUS = 'OLD'
 AND F.DATE < :CUT-DATE
 AND F.TYPE != 'RETRO';
```

```
DELETE FROM OFFER F
  USING PRODUCT PR
  WHERE F.TYPE = PR.TYPE
 AND PR.CLASS = 'REPLACED';
```

3.5 INSERT Command

3.5.1 Syntax

The INSERT command adds rows to an external-schema table.
The INSERT command has the following syntax:

INSERT INTO table-name (column-name ...)

```
VALUES { FROM { FILE 'file-name' }  
        { FILE ':variable-name' }  
        { STRUCTURE :variable-name }  
        value ... } ;
```

where

file-name and variable-name are defined in the host program,

column-spec is defined for the relational view,

table-label is a one- or two-character name,

value is a scalar variable a quoted variable, a number in the host program, or a character string.

column-spec is { column-name }
 { table-name.column-name }
 { table-label.column-name }

3.52 Comments

(a) Locking

An INSERT command issued inside a transaction usually places an EXCLUSIVE lock automatically (on rows or on tables, depending on the particular internal-schema database managers) until a COMMIT command is encountered.

(b) Specified Columns

The columns of the table are specified in the column-list. Values are supplied either from an external file, in which case many rows may be created, or from a source-list or data structure, in which case one row is created for each set of values. The values are related to columns in the column-list by their respective orders of appearance. The columns in the column-list need not be specified in the same order as the columns in the external-schema table were initially described to the system.

(c) File Input

If the values to be inserted are taken from a file, then multiple records can be inserted. The specification of file input causes an implicit loop to be generated that repeatedly executes the INSERT command until the file is empty. The file is read as string input. Each row is a record; the end of the rows is marked by the end-of-file. There are no delimiters between fields. It is assumed that the record format matches the format of the column list. The file-name is a logical file-name which should be related to a physical file through the system's job control language. The input file must be defined in the application program (by COBOL SELECT and FD statements).

If you have embedded the NDML statement in a FORTRAN program, the following rules apply. If inserting a character or integer value, the exact size of the external data item being inserted must be allotted in each input record. If a floating point value is being inserted, the exact size of the data item being inserted plus one extra space for the decimal point must be allocated. The actual decimal point must be included in the number if a floating point number is inserted. Character values must be left-justified in their space allocated. Floating point numbers must be both preceded and followed by the appropriate number of zeros to fill up the allocated space. Integer numbers must be preceded by the appropriate number of zeros to fill up the allocated space.

(d) Structure Input

The format of a data structure must match the format of the column list. It is assumed that the data type of structure fields exactly match that of the corresponding table columns in the external-schema format. Only one row can be inserted by this method without explicitly placing the NDML command within a program loop. Structure Input is not applicable if embedding the NDML statement in FORTRAN because structures as such do not exist.

(e) Value and Variable Input

A source list enclosed in parentheses can contain values and/or program variables for input. Multiple source-lists can be specified to cause an implicit loop to be generated that executes the INSERT command once for each source list. The data types of values explicitly given must agree with the data type of target columns. In this release, values cannot be calculated by an arithmetic expression within the INSERT statement.

If you have embedded the NDML statement in a FORTRAN program and you are inserting from variables, the following rules apply. If a character value is being inserted, the insert variable must be defined as CHARACTER *n, where n is the exact size of the external data item into which it is being inserted. If an integer value is being inserted, the insert variable must be defined as an INTEGER. If a floating point value is being inserted, the insert variable must be defined as DOUBLE PRECISION.

(f) Mapping Restrictions

The external-schema table (your view of the table) must map to one complete conceptual schema entity class. This means that a request to INSERT a row in a table in your view can be rejected by the system. Thus, it may be necessary to determine the conceptual-schema structure and mapping to external views to formulate a correct INSERT command to explicitly insert all the columns of a row in the conceptual schema.

The entity class (conceptual schema) may map to just part (or all) of one or more internal-schema (actual databases) record types. If just part of a record type is "mapped to," that part not inserted is filled with null-values. Moreover, if a record type in the internal database maps to two conceptual-schema entity classes, inserting in one conceptual entity, followed by the other, will result in two partial record instances in the internal database, rather than one complete instance; the Precompiler does not view this result as incorrect and will not issue a rejection or warning.

The NDML precompiler will generate update transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be updated. The update of these secondary copies are not guaranteed.

Furthermore, the precompiler will not reject multiple subtransactions being generated by the update action, one subtransaction per copy of data. The precompiler will continue to reject cases, where for a specified preference, one entity class maps to a non-normalized database structure resulting in multiple subtransactions.

(g) Integrity Constraints

A request to insert a conceptual-schema entity that is dependent in a relation class but for which no independent entity exists will be rejected at runtime. A dependent entity cannot exist without its associated independent entities, one for each relation class in which it is dependent.

A request to insert a conceptual-schema entity with key value equal to that of an entity already in the database will be rejected at runtime. Key values must be unique.

(h) Null Values

The specification of internal-schema null-values is DBMS dependent. The null values stored on the CDM for database by the CDMA will be used.

(i) WHERE clause of CREATE VIEW

It is permissible to insert rows not in the external view. Therefore, the qualifications in the WHERE clause of the CREATE VIEW will have no significance for NDML processing.

(j) Examples:

```
INSERT INTO DEPT
(DNO DNAME DLOC DSIZE)
VALUES FROM DEPT-FILE;
```

```
INSERT INTO DEPT
(DNO DNAME DLOC DSIZE)
VALUES (12 'ENGR' 'B1' 'SMALL');
```

```
INSERT INTO DEPT
(DNO DNAME DLOC DSIZE)
VALUES      (12 'ENGR' 'B1' 'SMALL')
            (40 'CUST' 'F4' 'SMALL')
            (36 'SW' 'G2' 'LARGE');
```

```
INSERT INTO DEPT
(DNO DNAME DLOC DSIZE)
VALUES      (:DEPT-NUM :DEPT-NAME 'B1' :DEPT-SIZE);
```

```
INSERT INTO DEPT
(DNO DNAME DLOC DSIZE)
VALUES FROM STRUCTURE :DEPT-REC;
```

where DEPT-REC has the structure:

```
01 DEPT-REC.
    03 DEPT-NUM          PIC 99.
    03 DEPT-NAME         PIC X(4).
    03 DEPT-LOC          PIC XX.
    03 DEPT-SIZE         PIC X(5).
```

3.6 MODIFY Command

3.6.1 Syntax

The MODIFY command changes values in an external-schema table. The MODIFY command has the following syntax:

```
MODIFY table-name [table-label]
      [USING table-name [table-label], ...]
      SET column-spec = value ...
      WHERE      { ALL                      }
                  { predicate-spec          }
```

where

table-label is a one- or two-character name,

table-name and column-spec are defined for the relational view,

value is a scalar variable, or a quoted variable, or a number in the host program

column-spec is: { column-name }
 { table-name.column-name }
 { table-label.column-name }

predicate-spec is either a column-, join-, between-, or a null-predicate (see Appendix A).

The columns to be changed and the values to be entered must be explicitly specified in the SET clause; values cannot be read from a structure or file.

3.6.2 Comments

(a) Integrity Constraints and Mapping Restrictions

Three specific integrity constraints are enforced by the system. First, the MODIFY command cannot be used to change the values of a column that corresponds to the key class of an entity class in the conceptual schema. Thus, some requests that have an apparently correct syntax might be rejected. To modify a key class, it is necessary to first DELETE and then INSERT the entity. Second, referential integrity is enforced. If a foreign key class is to be modified, there must exist a parent for the new key. Third, it is not permissible to change just part of a foreign key class; the entire foreign key must be changed.

Some columns cannot be modified alone because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. You can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

The NDML precompiler will generate update transactions for secondary copies if the CDMA has permitted it through the use of the ALLOW UPDATE clause of the CREATE MAP command. If DISALLOW UPDATE has been specified, which is the default, only the primary copy will be updated. The update of these secondary copies are not guaranteed.

Furthermore, the precompiler will not reject multiple subtransactions being generated by the update action, one subtransaction per copy of data. The precompiler will continue to reject cases, where for a specified preference, one entity class maps to a non-normalized database structure resulting in multiple subtransactions.

(b) Locking

A MODIFY command within a transaction usually places an EXCLUSIVE lock automatically (on rows or on tables accessed, depending on the particular internal-schema database managers) until a COMMIT command is encountered. A MODIFY command issued outside of a transaction usually commits the result immediately. The specific lock used is determined by the particular internal-schema database manager.

(c) USING Clause

The USING clause specifies tables that are accessed by the WHERE clause to qualify the request. These tables need not necessarily include the one that is being modified. To be meaningful, tables indicated in the USING clause must be related to the table on which the MODIFY command acts by a join-predicate.

(d) SET Clause

The SET clause specifies the new values that are to be given to values in designated columns. The new value can be contained in a program variable or be given explicitly. In this release, new values cannot be calculated by arithmetic expressions in the MODIFY command, nor can they be contained in a structure or file.

(e) WHERE Clause

The WHERE clause is mandatory. The WHERE clause is used to specify which rows qualify to be changed. If all the rows of a table are to be modified, then the WHERE ALL clause should be used. For selective qualification of rows, the WHERE clause has the same power of expression as it does in a SELECT statement. If the WHERE clause is not included in a MODIFY statement, the Precompiler will reject the statement and issue an error code. Because distributed update is not supported, the WHERE clause mapping to the multiple subtransactions per preference is not supported, since the query results of one subtransaction dictates the actual rows to be modified by another transaction.

The qualifications specified in the WHERE clause of an NDML statement will be "ANDed" with those specified in the WHERE clause of the CREATE view. These qualifications include the column to value predicates and may be expressed within nested parentheses. This parenthesized logic will be enforced by the Precompiler. The NDML precompiler will ignore the WHERE ALL qualification of the NDML statement when CREATE VIEW qualifications exist.

Note: It is permissible to modify rows in the view that are moved thereby out of the view.

Some columns cannot be specified in a WHERE clause because the column in the conceptual schema maps to non-normalized database structures in the internal-schema databases. In particular, a conceptual-schema column that maps to a data field in a repeating group in the internal database will not have a unique value for each row. The Precompiler should recognize this problem and reject the NDML request. The user can determine these restrictions before precompilation only by examining conceptual-internal schema mapping relationships.

(f) Examples:

```
MODIFY OFFER F
  SET F.STATUS = 'EXPIRED'
  WHERE F.DATE < :CUTDATE;
```

```
MODIFY OFFER F
  SET F.RESPONSIBLE-DEPT = 'BENEFITS'
  WHERE ALL;
```

```
MODIFY DEPT D
  USING EMPLOYEE EMP
  SET D.STATUS = 'INACTIVE'
  WHERE D.DNO != EMP.DNO;
```

```
MODIFY DEPT D
  SET D.STATUS = 'INACTIVE'
  D.LOC = 'INACTIVE'
  D.RESPONSIBLE-MNGR = :MNGR-INPUT
  WHERE D.DNO = :DEPT-NO-INPUT;
```

3.7 Transaction Commands

3.7.1 BEGIN TRANSACTION Command

The BEGIN TRANSACTION command indicates the start of one or a group of NDML commands that must be completed successfully as a unit in order to maintain the integrity of the database system. All automatic locks issued (for SELECT, INSERT, DELETE and MODIFY commands) and an explicit EXCLUSIVE lock placed by a SELECT command refer to this transaction. If locks exist from prior commands for an open transaction that have not been removed by a preceding commit-command or rollback-command, the BEGIN TRANSACTION command will issue a rollback-command to undo any uncommitted previous commands.

A transaction ends at the next UNDO, ROLLBACK or COMMIT statement. Transactions cannot be nested.

3.7.2 UNDO and ROLLBACK Commands

These NDML commands cause the system to undo any actions accomplished since the last BEGIN TRANSACTION command. The databases will be returned to their previous states.

3.7.3 COMMIT Command

The COMMIT command causes all actions accomplished since the last BEGIN TRANSACTION command to become permanent and all existing locks on records for this transaction to be removed. The following is an example use of the COMMIT command:

```
## BEGIN TRANSACTION;
## MODIFY OFFER F
##   SET F.RESPONSIBLE-DEPT = 'BENEFITS'
##   WHERE ALL;
##   IF NDML-STATUS = 'ERROR'
## ROLLBACK;
##   ELSE
## COMMIT;
```

3.8 Loop Construct

3.8.1 When a Loop Construct Is Needed

The host language compiler expects that all input and output in an application program be done a record at a time. In contrast, a single NDML SELECT command can return many records. The loop construct is provided to allow NDML to interact with the application program one record at a time.

A loop construct is necessary for assignment of multiple returned values to program variables or to a structure, even if the variables or structure fields are vectors. The major reason that implicit looping is not generated is that there is no way to determine the number of records to be returned during the precompile step; therefore, the programmer should test the number of records returned within an NDML loop construct to ensure that storage dimensions of the variables are not exceeded during execution.

It is not necessary to use a loop construct if only the first record returned is to be used. For example, a loop construct will never be necessary when functions are specified in a SELECT because only one row is returned. Specification is used because looping is implicit (the file is assumed to be capable of growing to hold all output).

Note that the loop executes after the SELECT retrieval is complete. Therefore, changing values in the WHERE or ORDER BY clauses within the loop will have no effect on the result.

3.8.2 Syntax

A loop must immediately follow a SELECT command. If a loop construct follows, do not end the SELECT command with a ";" because the end of the NDML procedure is indicated by the closing bracket. The start of the loop is indicated by "{" and the end by "}", both of which are embedded NDML statements and must be preceded on the line in the application program by appropriate NDML designation characters. The body of the loop can contain both host-language statements and embedded NDML commands.

It is permissible to include NDML statements within loop constructs for a SELECT statement. A transaction defined by a BEGIN TRANSACTION statement must either enclose the entire SELECT statement and associated loop construct or must be contained within the loop construct. An example of the latter is given under 3.7.4.

The following two restrictions on the use of loop constructs are important. Programmers should not attempt to exit a loop by using a host language GOTO or equivalent statement. The result of such a jump is undefined. Secondly, the NDML commands SELECT, INSERT, DELETE and MODIFY should not appear within a host-language "IF" statement because the Precompiler will not be able to guarantee the integrity of the logic path. The NDML statements listed under 3.7.4 are provided to control the processing of loops. (The NDML commands COMMIT, UNDO and ROLLBACK can also be placed within a host-language IF statement).

3.8.3 NDML Loop Control Statements

(a) CONTINUE or NEXT

This statement causes the current iteration of the loop to terminate and the next iteration to be generated. The NDML statement CONTINUE should not be confused with the FORTRAN statement.

(b) BREAK or EXIT

This statement causes the loop to be terminated and control to be passed to the program statement following the end of the loop.

3.8.4 Evaluation

The following actions are taken by the system to evaluate an embedded NDML SELECT statement:

1. The system evaluates the query and stores the resulting rows in a result file. If a file name has been specified by the programmer in an INTO phrase to receive the results, the result file is given the specified name and the command is finished. Otherwise, proceed.
2. The code within the loop specified in the SELECT command is executed, once for each row generated by the query. Values are moved to the program variables or structure fields specified to receive them. It is necessary that the host language code either move those values to safe storage or specify new variables (for example, new indices of array variables) for each execution of the loop if more than one row is returned. The host language code should also test the number of loops to ensure that the allocated storage for returned information is not exceeded.

The following example illustrates how program variables that receive information from a SELECT statement can be manipulated in a loop construct (this and the following examples are COBOL). Note that the braces should be on a separate line without following code.

```
## SELECT :PART-NUMBER = P.PARTNO,  
## :PART-NAME = P.NAME  
## FROM PARTS P  
## {  
##     DISPLAY PART-NUMBER, PART-NAME  
##     COMPUTE NUMBER-OF-PARTS = NUMBER-OF-PARTS + 1.  
## }
```

The following example shows how a COBOL variable can be used in the WHERE clause and how the CONTINUE statement can be used. Parts with a null part name are skipped. Otherwise, counters are incremented depending on the value of the work number.

```
## SELECT :PART-NAME = P.NAME, :WORK-NO = P.WORKNO  
## FROM PARTS P  
## WHERE P.SIZE = 'SMALL'  
## AND P.PARTTYPE = :PART-TYPE  
## {  
##     IF PART-NAME = SPACES  
##         CONTINUE  
##     IF WORK-NO < BREAK-POINT  
##         ADD 1 TO ODD-LOT-COUNT  
##     ELSE  
##         ADD 1 TO REGULAR-LOT COUNT.  
## }
```

The following example shows the inclusion of a transaction within a loop construct.

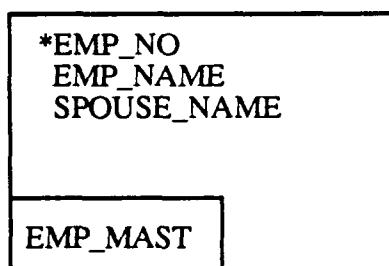
```
## SELECT :PART-NAME = P.NAME, :PART-COLOR = P.COLOR  
## FROM PARTS P  
## {  
##     BEGIN TRANSACTION;  
##     INSERT INTO COLORTABLE (CNAME CCOLOR)  
##     VALUES (:PART-NAME :PART-COLOR);  
##     IF NDML-STATUS = 'ERROR'  
##         ROLLBACK;  
##     ELSE  
##         COMMIT;  
## }
```

3.9 Distributed Update Restrictions

For examples 1 through 5, assume that the CDMA has "allowed" update for entity EMP-MAST. Also, there is a 1 to 1 mapping for AUCs EMP-NAME and SPOUSE-NAME, whereas the keyed AUC EMP-NO is mapped for preference 1 to database:1 and mapped for preference 2 to database:2

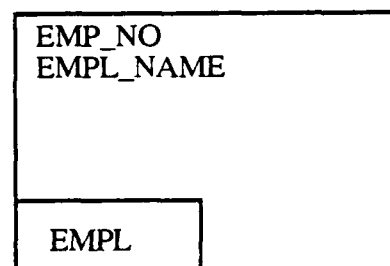
Conceptual Schema

(Assume EXTERNAL SCHEMA
is the same)

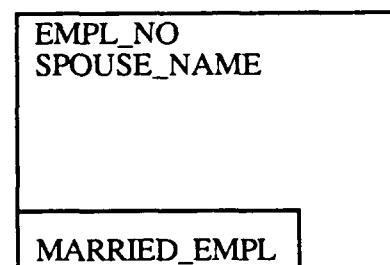


Internal Schema

DATABASE: 1



DATABASE: 2



NDML Transaction Examples:

1. INSERT INTO EMP_MAST
(EMP-NO EMP-NAME SPOUSE-NAME) VALUES
(100 'MR X' 'MRS X');
2. MODIFY EMP_MAST
SET SPOUSE_NAME = 'NEW MRS X'
WHERE EMP_NO = 100 ;

3. MODIFY EMP_MAST
 SET SPOUSE_NAME = 'NEW MRS X'
 WHERE EMP_NAME = 'MR X' ;
4. DELETE FROM EMP_MAST
 WHERE EMP_NO = 100 ;
5. DELETE FROM EMP_MAST
 WHERE EMP_NAME = 'MR X' OR
 SPOUSE_NAME = 'MRS X' ;

Example 1:

Two subtransactions will be generated to insert into records EMPL of Database:1 and MARRIED_EMPL of Database: 2.

There are no restrictions for "insert" Actions. All copies or sources will be updated.

Example 2:

1 subtransaction will be generated to modify record MARRIED_EMPL of Database:2 with the appropriate qualifications (i.e., where EMPL_NO = 100)

Example 3:

This NDML request will be rejected because the record we are attempting to update (MARRIED_EMPL of Database:2) does not contain the relevant qualification (i.e., EMP_NAME = 'MR X').

Example 4:

Two subtransactions will be generated to delete records EMPL of Database:1 and MARRIED_EMPL of Database:2. The CS to IS Transformer Module (where CS is conceptual schema and IS is internal schema) will recognize that the qualification criteria is present in all the records being deleted.

Example 5:

This NDML Delete will fail because both the qualifications are not present in both the records being deleted.

3.10 Error Codes

Error code values are defined by the IISS error handling philosophy. NTM and communication system errors are returned to the NDML application. Other codes of interest to the NDML application are:

49901 - failure of a type 1 referential integrity test on an insert or modify

49902 - failure of a type 2 referential integrity test on a delete

49903 - failure of a key uniqueness test on an insert

44306 - failure of a domain verification module

This error code will be found in the variable NDML-STATUS (or NDMLST in FORTRAN) after every NDML statement. This is a variable generated into the user program.

Note: A referential of an empty set is not considered as an error.

SECTION 4

NDML PRECOMPILE OVERVIEW

The IISS Precompiler will precompile your application process containing embedded NDML commands. The Precompiler parses the application program source code and identifies the NDML commands. It will modify the original application process to include numerous variables and subroutine calls necessary to implement the NDML commands in the host language. The Precompiler will generate code (generated query processes) that will be activated at run time to access the identified internal-schema databases and to perform the required internal-schema to conceptual-schema transforms. It will also generate code (generated conceptual/external transformer) that will be activated at run time to perform the required conceptual to external transforms, statistics functions, ordering of results, and other processes necessary to present the requested results to the application process.

In order to activate the IISS Precompiler, the procedure file GENAP must be used. This procedure file enables you to use the IISS Interface Application Generator (GAP), the IISS Precompiler, the IISS RP-Main Generator, and the required link/load options.

APPENDIX A

BNF OF THE NDML

A.1 Conventions

A.1.1 Notation

Certain conventions are used to describe the form of command

UPPER CASE WORDS denote keywords in the command

LOWER CASE WORDS denote user-defined words

{ } denotes that exactly one of the options within the braces must be selected by the user

"{" or "}" denotes a literal brace character without special meaning

[] denotes that the entry within the brackets is optional

| denotes an "or" relationship among the entries

_ denotes default option

A.1.2 Punctuation

1. A "." is used to separate the table-label (i.e., table alias) from the column-name. The table-label is used to match a column to a specific table in the list of tables referenced in the FROM clause.
2. A ":" is placed before the name of a host-language program variable, structure or file name that will receive returned values.
3. A "," is inserted between entries in the list of tables in a FROM clause.
4. A "," is inserted between subscripts to an array variable.
5. A set of parentheses is used to enclose the column-list in an INSERT statement.
6. A set of parentheses is used to enclose the object column of a function.
7. A set of parentheses is used to enclose the values to be inserted in an INSERT statement.
8. A set of parentheses is used to enclose a program variable subscript list.

9. A mandatory ";" or loop construct is placed at the end of the command.
10. A set of parentheses of group logical conditions in the WHERE clause. They may be nested.
11. A set of parentheses may be used to group combinations of SELECT statements. They may be nested.

A.2 NDML Backus-Normal Form (BNF)

ndml-command	::= select-command insert-command delete-command modify-command begin-recoverable-unit-command commit-command rollback-command query-expression
select-command	::= SELECT [lock-request] [INTO external-struct] [DISTINCT] {[table-label] ALL expr-list var-assgnmt-list} FROM table-list [WHERE predicate-list] [ORDER BY order-spec-list] {; loop construct }
insert-command	::= INSERT INTO table-name (column-list) VALUES {FROM external-struct source-list};
delete-command	::= DELETE FROM table-name [table-label] [USING table-list] WHERE {ALL predicate-list};
modify-command	::= MODIFY table-name [table-label] [USING table-list] SET column-assgnmt-list WHERE {ALL predicate-list};
query-spec	::= SELECT [DISTINCT] column-list FROM table list [WHERE predicate-list];

query-expression	::= SELECT { INTO external-struct scalar-variable-list function list } FROM (query-combination) [ORDER by scalar-variable-list]
query-combination	::= query-spec query-combination set-operator query-combination (query-combination)
set-operator	::= UNION INTERSECT DIFFERENCE
begin-recoverable unit-command	::= BEGIN TRANSACTION;
commit-command	::= COMMIT;
rollback-command	::= UNDO; ROLLBACK;
bool-op	::= = != > >= < <= <>
column-assignment-list	::= column-assgnmt-spec column-assgnmt-list column-assgnmt-spec
column-assignment-spec	::= column-spec = value
column-list	::= column-spec column-list column-spec
column-predicate	::= column-spec bool-op value value bool-op column-spec
column-spec	::= column-name table-name.column- name table-label.column-name
digit	::= 0 1 2 3 4 5 6 7 8 9
direction	::= ASC DESC ASCENDING DESCENDING UP DOWN
expr-list	::= expr-spec expr-list expr-spec

expr-spec	::= column-spec function([DISTINCT] column-spec)
external-struct	::= 'file-name' 'variable-name' STRUCTURE :variable-name
function	::= AVG MEAN MAX MIN SUM COUNT
function-list	::= function-spec function-list function-spec
function-spec	::= scalar-variable = function
integer	::= digit integer digit
join-op	::= = !=
join-predicate	::= column-spec join-op column-spec
lock-request	::= WITH [EXCLUSIVE SHARED NO] LOCK
loop construct	::= "{" statement-list "}"
number	::= integer [.[integer]]
order-spec-list	::= column-spec [direction] order-spec-list column-spec [direction]
predicate-list	::= boolean-term predicate-list {OR XOR} boolean-term
boolean-term	::= boolean-factor boolean-term AND boolean-factor
boolean-factor	::= [NOT] boolean-primary
boolean-primary	::= predicate-spec (predicate-list)
predicate-spec	::= column join-predicate between-predicate null-predicate
quoted-variable	::= 'literal-string'

scalar-variable	::= :variable-name [(subscript-list)]
source-list	::= (value-list)
scalar-variable-list	::= scalar-spec scalar-variable-list scalar spec
scalar-spec	::= scalar-variable
between-predicate	::= column-spec [NOT] BETWEEN {column-spec value} AND {column-spec value}
null-predicate	::= column-spec is [NOT] NULL
statement	::= host-language-statement ndml-command BREAK EXIT CONTINUE NEXT
statement-list	::= statement statement-list statement
subscript-list	::= integer subscript-list , integer
table-list	::= table-name [table-label] table- list, table-name [table-label]
value	::= scalar-variable quoted-variable number
value-list	::= value value-list value
var-assgnmt-list	::= var-assgnmt-spec var-assgnmt-list var-assgnmt-spec
var-assgnmt-spec	::= scalar-variable = expr-spec